Available online at www.sciencedirect.com
ScienceDirect
Procedia Computer Science 89 (2016) 473 – 482
Procedia Computer Science

Twelfth International Multi-Conference on Information Processing-2016 (IMCIP-2016)

Implementation of Interactive Real Time Online Co-Shopping using Push AJAX

Bhaskar Kumar[a,*], Kumar Abhishek[b], Akshay Deepak[b] and M. P. Singh[b]

[a] *EMS Telecom, Sonus Network, Bangalore, India*
[b] *National Institute of Technology, Patna, India*

Abstract

Online shopping has witnessed explosive growth: there are hundreds of millions of online shoppers that are part of an online retail industry worth of hundreds of billions of dollars. While online shopping has its own advantages (such as convenience, product reviews, price and selection range), recent focus has been in using technology to mimic the social interactions found in physical shopping malls and stores in what is called as "social shopping". Put simply, in social shopping shoppers' friends become involved in the shopping experience. An important aspect of social shopping is "co-shopping" where ad-hoc collaborative shopping groups are formed in which one person can drive an online shopping experience for one or more other people. This requires a mechanism that provisions multiple users to conduct real time online shopping collaboratively from multiple locations. For example, family members at remote locations can view the product online simultaneously as if they are shopping together in real time. To this end we have designed a special collaborative browsing window in which all the co-shoppers need to login. One user becomes the leader and invites others from his friends' list for shopping. Other users need to accept this invitation and join the shopping. The leader upon liking a product initiates the co-shopping by clicking on "Share this product with selected friends" button. On clicking this button an AJAX request, which contains URL of the product and list of IDs of selected friends, goes to the server. The server receives the URL of the product and list of IDs of friends and pushes this URL to all the selected friends in an asynchronous manner using PUSH AJAX technology. This URL gets loaded into an HTML iFrame on every client's browsing window. Shoppers can communicate with each other using a simple chat window.

Peer-review under responsibility of organizing committee of the Organizing Committee of IMCIP-2016

Keywords: AJAX; HTML; JSF; LONG POLL; PUSH AJAX; URL.

1. Introduction

E-commerce and wide penetration of internet has changed the way we shop today. The number of shoppers coming online for retailing has reached hundreds of millions and continues to grow. As per a report by China Internet Network Information Center[#], the number of users shopping online in China had reached 242 million in 2012 itself. The same figure for USA stands at 202 million in 2015[†] While India has the second largest internet base of about 354 million

*Corresponding author. Tel.: +91 -9901728973
E-mail address: bhaskar.mit2007@gmail.com
[#] http://news.xinhuanet.com/english/sci/2013-01/15/c_ 132104473.htm.
[†] http://www.statista.com/statistics/183755/number-of-us-internet-shoppers-since-2009/.

Peer-review under responsibility of organizing committee of the Organizing Committee of IMCIP-2016
doi:10.1016/j.procs.2016.06.108

users as of June 2015[‡], the penetration of e-commerce is low as compared to developed countries. However, the number of online shoppers is growing at an explosive rate of 6 million new entrants every month[§] and is expected to cross 100 million by 2016. It is not just the number of online shoppers running into hundreds of millions, but also the online retailing is running into hundreds of billions of dollars. For instance, the online spending in USA was around 307 billion dollars in 2014[**] and the same figure for China is around 300 billion dollars for 2013[††]. India is catching up fast and its e-commerce market is estimated to grow at a compounded annual growth rate (CAGR) of 63% to reach 8.5 billion dollars in 2016[‡‡].

1.1 Need for this design

Shopping is a social activity in which a group of people view the product and discuss about the product before buying it. Discussions and reviews make the buyer more comfortable in buying the product. The same ideology applies to Online Shopping as well. However, till now no good work exists that enables a customer to perform interactive real time co-shopping. All of the implemented works either use outdated technologies, raise performance issues, use open APIs such as Google Maps, or fail to serve in real time situations. More on the existing works on co-shopping are discussed in the "Related Works" section on the next page. Therefore, a method that can provision online co-shopping is required to make online shopping more real. Real time online co-shopping can be easily implemented using PUSH AJAX technology.

1.2 Push AJAX

HTTP supports unidirectional communication from client to server. It does not have provision for the "server-push" feature that notifies clients immediately if any change happens on server-side. Most designers today implement server-push by client-side polling, in which the client keeps on polling the server every fixed interval of time to get the latest server data. However, this design generates a lot of unnecessary traffic because clients keep on repeatedly polling the server even though there are no server side updates.

PUSH AJAX provides light weight, reliable and secure way to push notifications asynchronously from client to server. The reliable mechanism facilitates notifications delivery, fault-tolerant and high-availability supports. This part of implementation carries no application data, thus making it secure. Thus, real time online co-shopping can be easily implemented using PUSH AJAX technology.

PUSH AJAX uses a technique called Long Poll to update server data to clients. Once a client establishes the connection, the server gives an illusion to the client that more of data is still to come and holds back the connection. When the data is available the server sends the data through the held connection. At last, when the co-browsing ends, the server closes the connection. This way, real-time communication is achieved from server to client.

Long Poll technique has two significant disadvantages. Firstly, by default, browser allows only 2 connections per server. Since one connection is busy with PUSH AJAX, only one connection is remaining to service client's request. This may cause service delays. Also, uncontrollable factors like network delays cannot be handled and result in service delays.

In order to achieve AJAX PUSH in this article, ICEpush libraries and APIs are used. ICEpush provides lightweight asynchronous PUSH AJAX using long polling technology as described above. It provides two main jars to achieve PUSH AJAX – icefaces.jar and icepush.jar. In order to use ICEpush libraries, a proper well defined framework needs to be followed. ICEpush internally uses JSF framework that connects UI widgets with data sources and server-side event handlers. This is discussed in implementation part in detail.

[‡] http://articles.economictimes.indiatimes.com/2015-09-03/news/66178659_1_user-base-iamai-internet-and-mobile-association.

[§] http://timesofindia.indiatimes.com/tech/tech-news/Online-shoppers-in-India-to-cross-100-million-by-2016-Study/articleshow/45217773.cms?from=mdr.

[**] http://www.retailresearch.org/onlineretailing.php.

[††] http://www.advangent.com/2014/01/21/china-online-shopping-transaction-topped-rmb1-85-trillion-us300-billion-in-2013-growth-rate-stabilizing/.

[‡‡] http://www.assocham.org/newsdetail.php?id=5135.

1.3 Design summary

In summary, the online co-shopping works as follows:

a. leader user invites his online friends for co-shopping, (Offline friends cannot be invited.)
b. leader clicks on "Share this product with friends" button upon liking a product,
c. an AJAX request containing list of friends and URL of the product goes to the server,
d. the server manipulates the request by pushing the URL to all the friend's browsers.
e. Server pushes the URL into each client's HTML iFrame using PUSH AJAX.
f. Shoppers can communicate with each other using a chat window.

Figure 1 provides the pictorial representation of design summary by giving overview of steps.

2. Related Work

The growing E-commerce industry in online shopping attracted good amount of works on bridging the gap between buyers and sellers. However, very few works have been done on implementing an interactive real time co-shopping. In 2007, Khoury, Michel , Xiaojun Shen, Shirmohammadi, S proposed a "A Peer-to-Peer Collaborative Virtual Environment for E-Commerce". However, it fails to implement interactive realtime co-shopping with a group of friends. Moreover, it also relies on thirdparty VRML plugins for peer-to-peer communication to happen. In 2012, Jen-Hao Hsiao, Li-Jia Li proposed "On visual similarity based interactive product recommendation for online shopping". However, this design only provides a mechanism to make product recommendation method interactive based on similarity. It lacks implementation of real time shopping.

Richard Han, Veronique Perret, Mahmoud Naghshine in 2000 proposed a XML splitter based Multi-Device Collaborative Web Browsing. However, this design introduces a proxy to manipulate the requests, create and maintain sessions and split the XML pages. This proxy is an extra over head. Moreover, this design implements client side using an Applet based User Interface, which is hardly used these days. Applets are criticized for their Plug-In and JRE requirements, their startup performances and client side security issues. Modern browsers these days provide client server interface much better than what applets used to provide. Also, this approach is based on hidden client pull technology and assumes that universal push solution from server to client does not exist, which is a wrong assumption. AJAX PUSH is a technique that facilitates data push from server to clients when fresh data is available at the server.

Maria Aneiros, Vladimir Estivill-Castro (2005) proposed a Group Unified History (GUH) based collaborative web browsing. In this design, every time the user sends an HTTP request, a proxy server manipulates the request and passes it to GUH server. The GUH server then pushes all this information to the client updated history, and allows users to visit sites that others have judged or commented positively. However, this design is adds two extra servers – proxy and GUH server, thus making the design complex. Moreover, this design is specific to educational needs and would have to undergo lots of design changes to accomplish collaborative online shopping.

K. Maly, M. Zubair, and L. Li (2001) presented a design that removes the proxy server and describes an approach that supports a graphical Web browser with Java and Java Script. However, this design requires client side users to have JAVA enabled browsers and that form mutual exclusive groups of surfers that share the same experience. Moreover, this design again uses Applet based client side User Iinterface, which is outdated. Also, changes at the server are monitored through a Javascript Co-Pointer. The Co-pointer's relative position is monitored to measure the changes in the browsing window. This is an outdated technology because there are better mechanisms to measure the changes in the browsing window.

Alan W. Esenther 2002 presents a simple light weight web based real time collaborative web browser. Its instant messaging system provides a quick, simple and safe way to co-browse. Its' non-intrusive nature facilitates collaboration over existing web content without modification. However, this design cites some critical disadvantages and limitations. Pixel positions and resolutions of each user's browsing window needs to be same. So it is preferred to use the same browser brand on each client. Also, the server side co-browsing window logic needs to be placed in same domain as all web pages that are to be shared during the collaborative session.

Dietwig Lowet Daniel Goergen 2009 Co-Browsing Dynamic Web Pages, illustrates a magnificent way to implement co-browsing using Javascript DOM synchronization. However, this design requires Firefox configuration to allow

trusted JavaScript to receive cross domain scripting access and use the Firefox socket mechanism. This cross-domain scripting permission can lead to execution of unwanted Javascript and poses serious security issues. This is the reason, cross domain scripting permissions are disabled by default.

Jason J. Jung 2011, proposed a ContextGrid based contextual mashup-based collaborative browsing system in which a ContextGrid Model is used to compare user contexts. Also for better understanding of user context, open APIs such as Google Map APIs, are used to fetch additional information through various sources. This becomes a tedious approach to compare user contexts through various APIs to implement co-browsing because generating queries and filtering the query result and finding out relevant results will add to cost and complexity.

Raphael O. Santos, Roberta L. Gomes, Felip F. Oliveira, Magnos Martinello, 2011 proposed a magnificent design to achieve Lightweight Collaborative Web Browsing by developing OCEAN prototype. This design achieves co-browsing considering performance, responsiveness, scalability and user interface in an efficient way. Also, this architecture avoids bottlenecks on the server side, taking advantage of user's bandwidth in order to promote synchronous collaborative experiences. However, this design cites a concern related to feasibility of this design in real scenario due to unclear infrastructure. Moreover, this design is not suitable for dynamic web applications like online shopping.

Many research has happened in the field of co-browsing, however, most of them are based on master slave model, proxy based model or Javascript DOM synchronization based model, which are either outdated ways or complex ways to achieve server – client communication. There has been no model proposed with support for interactive realtime co-browsing, which is browser independent, simple to implement and light weight, a design that can facilitate requirements of realtime online shopping.

2.1 Comparison with Flipkart Ping App

India's E-Commerce giant Flipkart recently launched Ping, a new feature in its mobile app that allows users to chat with their friends while shopping, thus giving a start to a collaborative shopping experience. Ping allows users to have real-time conversations with friends in order to get feedback about a product they plan on buying, or help friends get in on the decision of what to buy. Using this feature, users can drag and drop products directly into conversations, share their screen, share products with friends from their cart, wish list, gallery and even allows them to take photos. However, this feature is limited to mobile phones without desktop support. Today, majority of shoppers prefer to shop on desktop for better visibility of product and their details. Online shopping on desktop gives users a wide interface for viewing the product details, its ratings and reviews, discount offers on various retailers and alternative options to the product that a shopping on mobile phone fails to give. Moreover, this app requires window change to go from chat window to product details window. Therefore, Flipkart's Ping is just a beginning to real time online collaborative shopping and the need for a desktop version of co-shopping still remains.

The implementation discussed in the paper, real time online collaborative shopping is achieved on desktop. All the product details and chat window are implemented into the same collaborative window and no window change is required. Using this desktop version of online co-shopping, users can have a better view of product details, its ratings and reviews, discount offers on various retailers and alternative options to the product.

3. Implementation of Interactive Real Time Online Co-Shopping

The implementation of this real time co-shopping design requires a client side collaborative window with HTML iFrames and an AJAX program, a server side PUSH AJAX implementation and a chat service. Client side collaborative window consists of three HTML frames. The first iFrame loads the pushed URL from the server and has provision to share the product URL with friends. The second frame has provision to invite users and third frame has a simple chat window. All the communications from client to server happen via AJAX routine.

The server side PUSH AJAX is implemented using AJAX PUSH APIs. AJAX PUSH empowers the server to push data from server to client and update any part of client's page at any time. AJAX PUSH can push message to a group of clients asynchronously. Collaborative multi-user applications for the web can be easily build using AJAX PUSH technique.

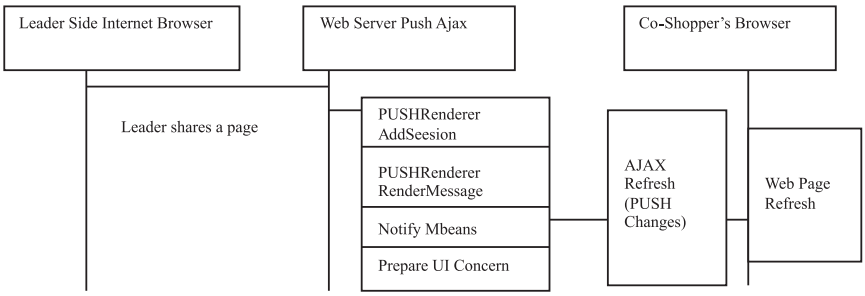

Fig. 1. Overview of Events in Collaborative Online Shopping.

All the users log in to the co-browsing window. After logging in, each user can see a user interface which consists of two frames as described above. Any one of the users can become a leader and initiate co-shopping by inviting his friends. Leader, upon liking a product, clicks on "Share this product with selected friends" button. An AJAX request goes to server. The AJAX request contains list of Ids of friends and URL of the product is sent to the server. The server manipulates the request by pushing the URL on the HTML frames of each co-shoppers in an asynchronous manner by using PUSH AJAX. Shoppers can communicate with each other using a chat window. Figure 1 shows the overview of events involved. Each of the implementations is discussed in detail.

3.1 Client side window and AJAX routine implementation

Client side collaborative window consists of an XHTML document. XHTML is HTML defined as an XML application and is supported on all major browsers. XHTML combines the strength of HTML and XML. Our client side XHTML document basically consists of three HTML iFrames. The first iFrame loads the pushed URL from the server. The second iFrame has provision to invite users, "share the product URL with friends" button and a hidden text input. The third iFrame has a chat window.

In order to implement the first frame, XML namespace for Java Server Faces, apart from xhtml namespace, must be included in the HTML tag as below. This is necessary because PUSH AJAX pushes message from server to client using Java Server Faces interfaces. Java Server Faces technology establishes the standard for building server-side user interfaces. JSF provides a rich control named DataTable to render and format html tables. Everytime server pushes the URL into this JSF datatable. JSF datatable provides onchange() method to perform desired action when updated data is pushed from the server. This onchange() method updates the iFrame src and reload iFrame by implemented updateiFrameSrc() method.

```
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:f="http://java.sun.com/jsf/core">
<h:body>
    <h:dataTable value="#{messageBean.urlString}" var="current"
        onchange="updateiFrameSrc
('#{current.text}')">
      <h:column>
          <h:outputText value="#{current.text}"/>
      </h:column>
    </h:dataTable>
</h:body>
```

The first iFrame always displays the URL pushed by the server's session, which is same as leader's web URL. The source of this iFrame keeps on changing and reloading as per the URL message pushed from the server. This frame handshakes with server side code MessageBean.java and fetches urlString everytime its changed. As discussed above,

the connection from server to client is kept open and server can handshake with client's JSF datatable. At the end, when the co-browsing ends, the server closes the connection.

The second iFrame consists of a , which contains list of friends. Leader selects some friends and clicks on "Invite selected friends" button. Doing so, a request containing list of friend's ID goes to the server and all these friends get included in the co-shopping. Leader upon liking a product clicks on "share the product URL with friends" button. Doing so, a request containing the product URL goes to the server. The server then pushes the URL to each client's browser.

The entire client to server communication is implemented by AJAX routine. The AJAX routine is written in a scripting language such as Javascript/Jquery. Leader, upon liking a product, clicks on "Share this product with selected friends" button. On this button click, an AJAX request, which contains list of Ids of friends and URL of the product, is sent to the server.

```
$("button").click(function(){
$.ajax({method:"get",
url: "/updateAllSelectedFriends?url="+window.location.href+"&friendsId="+listOfSelectedFriends,
success: function(result){
$("#div1").html(result);
}});});
```

The third iFrame simply consists of a chat server. Messages pushed from the server gets loaded into all the client's frame. This is just a simple chat server, which co-shoppers might use to discuss and reviews the product among themselves

3.2 Server side PUSH AJAX implementation

3.2.1 Overview of ICEPUSH

ICEPUSH is a technology that provides AJAX PUSH capabilities. ICEPUSH is based on light weight asynchronous notification core that leverages long polling over HTTP to enable AJAX PUSH.

The common component ICEpush Core implements an asynchronous notification mechanism that triggers client-side JavaScript logic from server-based triggers. As a pure notification mechanism, the Core delivers no payload specific to application. This is done by either an integration layer or application specific business logic layer. This approach provides scalability and security.

3.2.2 Notification core

The ICEpush core implements the asynchronous notification mechanism using long polling over HTTP. This mechanism provides light in weight, reliability and Security. The mechanism is purely for notification, making it extremely lightweight, and massively scalable. The reliable mechanism facilitates notifications delivery, fault-tolerant and high-availability supports. This part of implementation carries no application data, therefore it cannot be exploited.

Long polling needs to block connections to return notifications asynchronously to clients. The core handles all the responsibilities to block connections, like Connection Sharing and Connection Refresh. ICEpush makes sure that the browser only uses a single connection for notifications throughout the co-browsing session. The same connection is shared between cross windows, in case when multiple windows are connected to push enabled co-browsing web applications in the same domain.

The blocked connection becomes unreliable and can deteriorate over time, therefore that connection needs to be refreshed when idle. The ICEpush mechanism periodically ensures the blocked connection remains healthy.

3.2.3 Integration layer

The integration layer provides a low-level notification core to enable basic push capability to an application, integrated with the application. An integration layer provides the connection between the ICEPUSH core and

co-browsing web application. Integration required to support AJAX PUSH includes tools like Servlet, JSF and JSP. The ICEpush core mechanism provides a Servlet 3.0 implementation that handles all ICEpush related connections. ICEfaces also provides a Java Server Faces layer to integrate to the ICEpush core, and push capabilities through the ICEfaces Ajax Push APIs.

3.2.4 Programming model

PUSH AJAX can be easily implemented using ICEPUSH APIs provided in icepush.jar. The class PushRenderer provides functions to support PUSH AJAX in web applications to renderer in a JSF context. First of all, group management has to be done. All the client's sessions need to be added to a group using addCurrentSession () method. If a person exits from the co-browsing, his session needs to be removed using removeCurrentSession() method. Finally, when some notification happens at the server, PushRenderer.render (GROUP_NAME) can render the information from server to all the clients listed in the GROUP_NAME.

The request coming from the server contains product URL and list of friends attending the co-shopping. The job of the server is to send this URL to these set of friends asynchronously so that the URL gets loaded to each friend's HTML iframe. This can be implemented on the server side using a technology called PUSH AJAX. PUSH AJAX is a technology by which data can be pushed from server to client's browser in an asynchronous manner. PUSH AJAX on server can update any part of web page on the client's browser at any time. Actions of one user can be instantaneously communicated with other users sing PUSH AJAX. PUSH AJAX provides APIs to push messages from server to client.

The paper uses the PUSH AJAX APIs are used to achieve server to multi-client message push. The API class PushRenderer contains methods to add/remove clients session to PUSH AJAX and render the messages to all clients session. In order to use PushRenderer class methods, PUSH AJAX Framework structure needs to be followed. Below are the steps to follow framework structure to create a PUSH AJAX application:

1. First of all, a POJO class was created say TrackURL.java, which has a constructor and has a getter and a setter method for urlString. The method geUrlString() should return the value of urlString variable, which is the current URL of the leader. The method setUrlString(url) sets the variable urlString with current URL of the leader.
2. Then ApplicationScoped MessageBean was created as below:
   A file say MessageBean.java was created. Annotation @ManagedBean (name="messageBean") was used to configure Managed Beans named messageBean. Also, annotation @ApplicationScoped was used to specify that this bean application is scoped and the application scope will be active throughout our application life time. The file MessageBean.java contains an ArrayList of TrackURL class objects as well as getters and setters for this ArrayList. A client side method addToList (sessionId, url) keeps on adding to list of URLs visited by leader for each user. Whenever addToList() method adds a new URL, PushRenderer class render(PUSH_GROUP) pushes this URL to all client sessions registered under PUSH_GROUP.
3. Then a ViewScoped BroadcastMessageBean was created as below: (Viewscoped unlike ApplicationScoped, persists for a single user's interaction with a web application.)
   A file say BroadcastMessageBean.java was created.
4. AnnotationManagedBean(name="broadCastMessageBean") was used to configure Managed Beans named broadCastMessageBean. Also, annotation @ViewScoped was used to specify that one bean instance will be created for each client session. Then the annotation inside the BroadcastMessageBean class @ManagedProperty (value="#{messageBean}") instructs the system to inject a value into this property named messageBean. In the constructor of BroadcastMessageBean(), the session ID of current HTTP session for this client is tracked ,using FacesContext class as below. Also, each session of all associated pages (views) of a group are added using PushRenderer class addCurrentSession() method as below. API class javax.faces.context.FacesContext contains all of the per-request state information related to the processing of a single JavaServer Faces request, and the pushing of the corresponding response.

```
FacesContext facescontext = FacesContext.getCurrentInstance();
HttpSession session = (HttpSession)facescontext.getExternalContext().getSession(false);
sessionId = session.getId();
PushRenderer.addCurrentSession(PUSH_GROUP);
```

Finally, this class contains a client side method say updateURLOnEachClient(), which calls client side addToList (sessionId, url) method of MessageBean and adds URL to the list of visited sites by the leader user.

The static final string constant PUSH_GROUP, which represents our push group, is contained in file BroadcastMessageBean.java. Then all the current client session is added to PUSH_GROUP. Finally, when the trigger happens, PushRenderer class render(PUSH_GROUP) pushes this URL to all client sessions registered under PUSH_GROUP.

3.2.5 Chat support

To facilitate communication between all users, a short term solution of chat facility was added. There are many free ware JAVA chat box classes which provide this functionality. For this design FreeCS was chosen and integrated it into our Co-Browsing. FreeCS is a free chat server implemented in Java. It provides features like fully customised layout, SQL authentication or no authentication and authorization framework.

While co-shoppers are reviewing a product, they might need to communicate with each other. For this purpose, a simple chat server is implemented which broadcasts one member's message to every other member. A simple chat server can be implemented using the same PUSH AJAX technique described above. When a user types something and presses ENTER, a request, which contains typed message and list of Ids of co-shoppers, goes to the server. The server manipulates the request and pushes the message to all co-shoppers in the list. This message gets loaded into chat window of each client's browser frame.

The steps involved on the server side is described as below:

- Server receives the URL of the product that leader wants to share and list of friends participating in the co-shopping.
- An AJAX push event is triggered upon receiving this request.
- Using the PushRenderer class APIs, all the sessions of co-shoppers are added to PushRenderer current session.
- An implemented MessageBean class manages and maintains the list of URLs.
- Finally PushRenderer class render() method renders the URL into a JSF context. This URL goes to the client's browsing window and loads the URL into the first frame. The implemented class BroadcastMessage.java invokes this render() method. Since this class includes @ViewScoped and @ManagedProperty annotations, the AJAX PUSH renders this message to all clients.

4. Results and Discussion

The proposed design provides a system to perform realtime online co-shopping. The art provides an efficient platform where a group of shoppers can together view a product and discuss about it. Fig. 1 gives an overview of events that occur in the implemented design. Figure 2 and Fig. 3 shows the User Interfaces of leader side and co-shoppers side. A leader, upon liking a product, invites his friends for online co-shopping. All the co-shoppers accept this invitation. Leader then clicks on "Share this product with friends" button upon liking a product as shown in Fig. 2. An AJAX request containing list of friends and URL of the product goes to the server, the server manipulates the request by pushing the URL to all the friend's browsers. Server pushes the URL into each client's HTML frame using PUSH AJAX. All the co-shoppers can view the product selected by leader (Fig. 2). Shoppers can communicate with each other using a chat window as shown in Fig. 2 and Fig. 3.

PUSH AJAX pushes messages from server to a group of clients in asynchronous manner. Thus, there is no waiting time for one message to get delivered before other message. This saves over all time. All the APIs used in this implementation are packaged into **icepush.jar**. This jar is freely available. ICEpush is based on Servlet 3.0 standard ARP APIs.

4.1 Design evaluation

The proposed design for co-shopping meets an appropriate requirement. Though it might have some shortcomings, it will meet the primary requirements of the concern. Best of our effort have been put to make it a realistic design

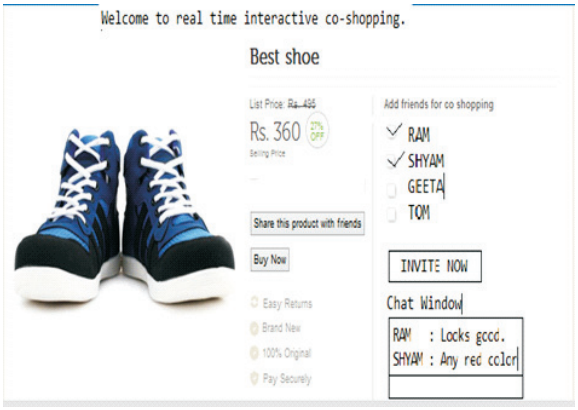

Fig. 2. Leader Side User Interface.

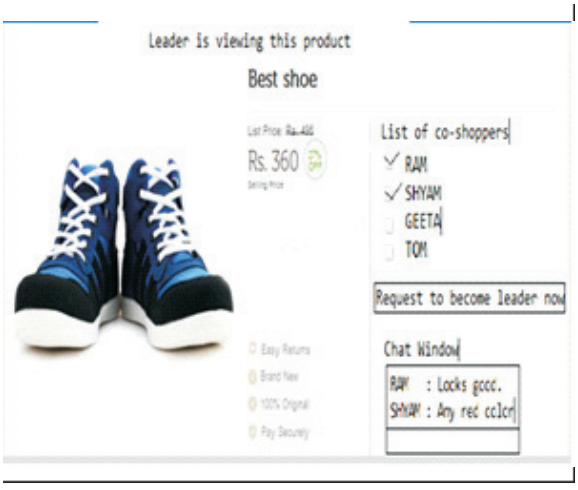

Fig. 3. Co-shoppers Side User Interface.

and more provisions can be added to it to meet any further requirements. The design can be extended by making modifications when the necessity arises in due course.

5. Conclusions

It can be easily concluded that the proposed system facilitates shoppers to shop with their friends and relatives in real time. The art provides an elegant design, both from client and server side to achieve its goal. It provides a flexible way for all the co-shoppers to shop together. This implementation is practically possible and can be deployed across various platforms.

6. Scope and Limitation

The proposed design also is subject to limitations of PUSH AJAX framework. Long Poll technique used in this design has two significant disadvantages. Firstly, by default, browser allows only 2 connections per server. Since one

connection is busy with PUSH AJAX, only one connection is remaining to service client's request. This may cause service delays. Also, uncontrollable factors like network delays cannot be handled and result in service delays.

The proposed design is subject to all kinds of network limitations. Deployment requires formatting the setup in a generic way so that it works for all different types of devices. As the number of shoppers increases, the server may suffer from overloading problems. Proper testing must be done to detect any memory leak issues on the server side.

7. Future Scope

The proposed work can be further enhanced to provide more privileges to leader. For example, a leader can terminate any of the co-shoppers from shopping. Another enhancement could be that any of the co-shoppers could request to become leader. Server side functionality can be further enhanced to provide security, privacy and load distribution. The AJAX script can be further enhanced to handle data validation, data editing, etc. Caching mechanism can be implemented such that list of co-shoppers need not be fetched from the server again on page refresh. Facility for recording the list of product viewed in the co-shopping and saving the chat history can be provided so that co-shoppers can review the shopping history. This idea can be enhanced to implement not only online shopping, but also areas where co-browsing is required, for example, real-time online game, real-time stock market watch, etc.

References

[1] R. Han, V. Perret and M. Naghshineh, WebSplitter: A Unified XML Framework for Multi-Device Collaborative Web Browsing, In *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*, ACM, pp. 221–230, December (2000).
[2] M. Aneiros and V. Estivill-Castro, Usability of Real-Time Unconstrained WWW-co-browsing for Educational Settings, In *The 2005 IEEE/WIC/ACM International Conference on Web Intelligence, 2005. Proceedings*, IEEE, pp. 105–111, September (2005).
[3] K. Maly, M. Zubair and L. Li, CoBrowser: Surfing the Web Using a Standard Browser, (2001).
[4] A. W. Esenther, Instant co-Browsing: Lightweight Real-Time Collaborative Web Browsing, In *Proc. of WWW*, (2002).
[5] D. Lowet and D. Goergen, Co-Browsing Dynamic Web Pages, In *Proceedings of the 18th International Conference on World Wide Web*, ACM, pp. 941–950, April (2009).
[6] J. J. Jung, ContextGrid: A Contextual Mashup-Based Collaborative Browsing System, *Information Systems Frontiers*, vol. 14(4), pp. 953–961, (2012).
[7] R. O. Santos, F. F. Oliveira, R. L. Gomes, M. Martinello and R. S. Guizzardi, Lightweight Collaborative Web Browsing, *Web Portal Design, Implementation, Integration, and Optimization*, vol. 17, (2013).
[8] M. Khoury, X. Shen and S. Shirmohammadi, A Peer-To-Peer Collaborative Virtual Environment for E-Commerce, In *CCECE 2007. Canadian Conference on Electrical and Computer Engineering, 2007*, IEEE, pp. 828–831, April (2007).
[9] J. H. Hsiao and L. J. Li, On Visual Similarity Based Interactive Product Recommendation for Online Shopping, In *2014 IEEE International Conference on Image Processing (ICIP)*, IEEE, pp. 3038–3041, April (2014).
[10] Flipkat Ping App – http://stories.flipkart.com/ping-flipkart-mobile-app/.
[11] ICEfaces Ajax Push – ICEsoft Technologies http://www.icesoft.org/java/projects/ICEfaces/ajax-push.jsf.
[12] ICEfaces Push Technologies – ICEsoft Technologies http://www.icesoft.org/java/projects/ICEfaces/push-technologies.jsf.
[13] Using Annotations to Configure Managed Beans – The Java EE 6 Tutorial http://docs.oracle.com/javaee/6/tutorial/doc/girch.html.
[14] ApplicationScoped (Java EE 6) – Oracle Documentation https://docs.oracle.com/javaee/6/api/javax/enterprise/context/ApplicationScoped.html.
[15] Ajax Push – APIs – ICEfaces – ICEfaces.org Community Wikihttp://www.icesoft.org/wiki/display/ICE/Ajax+Push+-+APIs.
[16] ManagedProperty (Java EE 6) – Oracle Documentation 7.https://docs.oracle.com/javaee/6/api/javax/faces/bean/ManagedProperty.html.
[17] FacesContext (Java EE 6) – Oracle Documentation https://docs.oracle.com/javaee/6/api/javax/faces/context/FacesContext.html.
[18] XHTML Tutorial – TutorialsPoint http://www.tutorialspoint.com/xhtml/.
[19] ICEpush – ICEsoft http://www.icesoft.org/java/projects/ICEpush/overview.jsf.
[20] JavaServer Faces Technology Overview http://www.oracle.com/technetwork/java/javaee/overview-140548.html.
[21] JavaServer Faces (JSF) Tutorial – TutorialsPoint http://www.tutorialspoint.com/jsf/.
[22] Jsf – Display Datatable – Tutorialspoint.comhttp://www.tutorialspoint.com/jsf/jsf_ display_ datatable.html.
[23] Shen and Jia, Social Comparison, Social Presence, and Enjoyment in the Acceptance of Social Shopping Websites, *Journal of Electronic Commerce Research*, vol. 13.3, p. 198, (2012).
[24] L. Zhou, P. Zhang and H. D. Zimmermann, Social Commerce Research: An Integrated View, *Electronic Commerce Research and Applications*, vol. 12.2, pp. 61–68, (2013).
[25] H. Zhang, Y. Lu, P. Gao and Z. Chen, Social Shopping Communities as an Emerging Business Model of Youth Entrepreneurship: Exploring the Effects of Website Characteristics, *International Journal of Technology Management*, vol. 66.4, pp. 319–345, (2014).